基于区块链的身份认证机制的效率优化方法研究 *

汤凌韬，许　敏，金玉荣

(江南计算技术研究所，江苏 无锡 214083)

摘　要：基于区块链的去中心化 PKI 使用区块链替代传统的 CA，借助区块链的权威性和公开性，实现了认证的去中心化。目前的研究大多使用遍历区块链的方法查询身份—公钥对，从而验证某公钥是否属于通信对方，效率较低。提出了一种基于密码累加器的身份认证方式，将链上身份和公钥信息映射为累加值，实现认证功能的同时提高了身份—公钥对的验证效率，同时解决了区块链体积不断增长的情况下轻节点存储空间不够的问题，并通过实验验证了该方法的可行性和有效性。

关键词：区块链；密码累加器；去中心化；身份认证

中图分类号：TP311　　doi: 10.3969/j.issn.1001-3695.2018.03.0211

Research on methods of improving efficiency of identity authentication based on blockchain

Tang Lingtao, Xu Min, Jin Yurong

(*Jiangnan Institute of Computing Technology*, *Wuxi Jiangsu* 214083, *China*)

Abstract: Decentralized public key infrastructure based on blockchain abandons certificate authority. With authoritativeness and publicity of the blockchain, this infrastructure is able to implement decentralized identity authentication. Many researches chose to traverse the entire blockchain to look up for a specific id-pk pair and then verify whether the public key belongs to someone who claims it. However, this method is obviously inefficient. This paper proposed an identity authentication method based on cryptographic accumulators, which map identity, public key and auxiliary information to one accumulated value. This method improved the authentication efficiency, especially when current blockchain was large. In addition, it solved the problem that lightweight clients do not had enough storage capacity when the size of blockchain was continuously increasing. Some experiments are also carry to measure this method and verify its feasibility and correctness.

Key words: blockchain; cryptographic accumulator; decentralization; identity authentication

0　引言

公钥基础设施(public key infrastructure,PKI)在实际应用中是一套软硬件系统和安全策略的集合，提供一整套安全机制，使用户在不知道对方身份或分布地很广的情况下，以证书为基础，通过一系列的信任关系进行通信和电子商务交易。作为目前网络安全建设的基础与核心，PKI 是电子商务安全开展的基本保障。一个传统的 PKI 系统包括证书机构 CA、PKI 策略、软硬件系统、注册机构 RA、证书发布系统和 PKI 应用等。由于传统的依托 CA 为中心的 PKI 在身份认证方面会产生单点失效、入侵目标明显等问题，催生了去中心化 PKI 的产生[1~4]。

区块链[5~7]技术自从 2008 年中本聪提出比特币后倍受关注，作为一种不可窜改的公开账本，被广泛用于在线支付、分布式存储和防伪证明等领域[18,19]。而去中心化 PKI 实现方法之一正是利用区块链存放身份和公钥等信息，在实现身份认证的过程中，采用遍历区块链的方法查询证书，再检查该公钥是否属于其所声明身份，最后发送挑战信息，通过验证数字签名判断对方是否持有匹配的私钥。然而区块链作为一条只可添加的公开链，其特性保证了数据量将不断增长，近几年区块链的体积已超过 100 Gb，并且未来仍会不停增长。届时，遍历区块链的方法效率将愈加低下，若仍沿用传统方法，身份认证的所需时间将难以满足实际需求；同时对于诸如手机等载体，无法存储如此庞大的数据，而密码累加器 (cryptographic accumulator) [8~16]的引入则可解决身份认证过程中成员验证效率低下的问题。密码累加器体制(accumulator scheme)包含一种算法，可将一个包含多元素的集合映射为一个累加值，并且提供一个体积较小的见证(witness)，证明一个给定的值确实属于该集合。

本文从上述现状出发，提出一种利用密码累加器实现去中心化 PKI 体制中身份认证的效率优化方法，主要研究密码累加器的定义、实现原理与特征对比，及其结合区块链应用于成员

收稿日期：2018-03-29；**修回日期：**2018-05-14　　**基金项目：**国家自然科学基金资助项目（91430214）

作者简介：汤凌韬（1994-），男，江苏启东人，硕士研究生，主要研究方向为网络安全（tangbdy@126.com）；许敏（1981-），男，工程师，硕士，主要研究方向为网络安全；金玉荣（1981-），男，助理研究员，硕士，主要研究方向为计算机安全.

验证的方法，提升身份认证的效率。

1 密码累加器

1.1 单向累加器

密码累加器的概念源于 Benaloh 等人[8]首次提出的单向累加器(one-way accumulators)，单向累加器被定义为一簇具有拟交换性(quasi-commutativeness)的单向哈希函数(one-way hash functions)，这里的单向哈希函数和常见的只有一个自变量的哈希函数不同，考虑接收两个自变量输入的情况。

定义 1 单向哈希函数。一簇单向哈希函数是满足下列条件的属于函数集 $\{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ 的无限序列，其中 $\mathcal{H}_\lambda \doteq \{h_k : X_k \times Y_k \to Z_k\}$ (k 为安全变量)：

a)对任何整数 λ ，以及任何 $h_k \in \mathcal{H}_\lambda$ ， $h_k(\cdot,\cdot)$ 在 λ 的多项式时间内可计算。

b) 对任何多项式时间的算法 $\mathcal{A}$ ，满足

$$Pr[h_k \xleftarrow{R} \mathcal{H}_\lambda; x \xleftarrow{R} X_k; y, y' \xleftarrow{R} Y_k; x' \leftarrow \mathcal{A}(1^\lambda, x, y, y'):$$
$$h_k(x,y) = h_k(x',y')] < \mathbf{negl}(\lambda)$$

其中：概率取决于 h_k, x, y, y' 的随机选择和 $\mathcal{A}$ 的随机输出。

由上述定义看出，单向哈希函数具有可计算性和单向性，即 x,y 给定，$z=h(x,y)$ 的计算可在多项式时间内完成；并若在给定 x,y,y' 的情况下，要找到 x' 满足 $h_k(x,y)=h_k(x',y')$ 的概率小到几乎可以忽略，即不同输入生成的输出间的冲突(collisions)很小。

定义 2 拟交换性。一个函数 $f: X\times Y \to X$ 被称为具有拟交换性，当下式成立：

$$(\forall x\in X)(\forall y_1,y_2\in Y)[f(f(x,y_1),y_2)=f(f(x,y_2),y_1)]$$

由此可见，具有拟交换性的函数，对同一个集合中的值依次进行哈希运算，在初始值不变的情况下，计算结果不随计算顺序的变化而改变。

单向累加器是具有拟交换性的单向函数，其可将一个包含多个元素的集合 L 映射为一个常数值 z，并为每个成员 $y\in L$ 提供一个证明，通过 y 和该证明可以得到累加值 z ，从而实现成员验证的功能。

1.2 强单向累加器

定义 1 表明了单向哈希函数在变量按规定选取时，找到冲突的概率极小。但未考虑攻击者可能不按指定范围取值，找到一个 $y'\notin Y$ 满足 $z=h(z_i,y')$ ，从而伪造凭证。为了增强安全性，引入强单向 (strongly one-wayness) [9]的定义。

定义 3 强单向哈希函数。

一簇强单向哈希函数是满足下列条件的属于函数集 $\{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ 的无限序列，其中 $\mathcal{H}_\lambda \doteq \{h_k : X_k \times Y_k \to Z_k\}$ (k 为安全变量)：

a)对任何整数 λ ，以及任何 $h_k \in \mathcal{H}_\lambda$ ， $h_k(\cdot,\cdot)$ 在 λ 的多项式时间内可计算。

b)对任何多项式时间的算法 $\mathcal{A}$ ，满足

$$Pr[h_k \xleftarrow{R} \mathcal{H}_\lambda; x \xleftarrow{R} X_k; y \xleftarrow{R} Y_k; (x',y') \leftarrow \mathcal{A}(1^\lambda, x, y):$$
$$y' \neq y \wedge h_k(x,y) = h_k(x',y')] < \mathbf{negl}(\lambda)$$

其中：概率取决于 h_k, x, y 的随机选择和 $\mathcal{A}$ 随机输出。

定义 1 中的单向性是指，给定值 $(y_1,y_2,\ldots,y_n)$ ，它们的累加值 z ，以及另一个值 y' ，攻击者难以找到对应见证 $accu'$ 使得 $h(accu',y')=z$ 。而强单向性是指，给定 $(y_1,y_2,\ldots,y_n)$ 和 z ，难以找到数值对 $(y',accu')$ 使得 $h(accu',y')=z$ ，并且 $y'\notin(y_1,y_2,\ldots,y_n)$ 。

1.3 无冲突累加器

Baric 等人[11]在构建 FSS 机制(fail-stop scheme)时提出了，密码累加器需要更严格的性质，在强单向的性质下，攻击者仍可能精心伪造成员值 $(y_1',y_2',\ldots,y_n')$ ，从而为 y' 构造见证 $accu'$ 。因此引入无冲突累加器(collision-free accumulator)，在强单向性基础上，成员值 $(y_1,y_2,\ldots,y_n)$ 无需给定。为了引入无冲突性，首先定义一般意义的累加器：

定义 4 累加器运行机制。一个密码累加器的运行机制[7]是一个包含 4 个多项式时间算法(Gen, Eval, Wit, Ver)的四元组，其中：

——密钥生成算法 Gen，一个用以生成初始参数的概率算法。Gen 接收两个参数：一个安全变量 1^λ ，一个累加器门限 N ，可被安全累加的值个数上限，当对超过 N 个元素的集合使用累加器，安全性则不能保证；返回一个累加器密钥 k ， $k\in\mathcal{K}_{\lambda,N}$ 。

——求值算法 Eval，一个求累加值的概率算法。计算集合 $L\doteq\{y_1,y_2,\ldots,y_{N'}\},N'\le N$ 中的所有值的累加值，其中 $y_i\in Y_k, k\in\mathcal{K}_{\lambda,N}$ 。Eval 输入 $(k,y_1,y_2,\ldots,y_{N'})$ ；输出一个累加值 $z\in Z_k$ ，和一些附加信息 aux，用做其他算法的输入。注意到 Eval 对相同输入输出同一个累加值，而附加信息可能不同。

——见证生成算法 Wit，一个根据相关信息生成成员见证的概率算法。Wit 输入一个累加器密钥 $k\in\mathcal{K}_{\lambda,N}$ ，一个值 $y_i\in Y_k$ 以及 Eval $(k,y_1,y_2,\ldots,y_{N'})$ 输出的附加信息 aux；若 y_i 确实在 L 中，输出一个见证 $w_i\in\mathcal{W}_k$ ，用于证明 y_i 被累计入 z ，否则，返回符号 ⊥ 。

——验证算法 Ver，一个通过见证来验证某值成员身份的确定性算法。Ver 输入 (k,y_i,w_i,z) ，根据见证 w_i 来验证 y_i 是否被累计入 z ，输出 Yes 或 No。

由一般累加器可进一步定义无冲突累加器。

定义 5 无冲突性。一个累加器机制被称为 N 次无冲突当其满足下述性质：对任意整数 λ 和任意多项式时间概率算法 $\mathcal{A}$ 满足

$$Pr[k \leftarrow \mathrm{Gen}(1^\lambda, N); (y_1,\ldots,y_N, y', w') \leftarrow \mathcal{A}(1^\lambda, N, k);$$
$$(z,\mathrm{aux}) \leftarrow \mathrm{Eval}(k, y_1,\ldots,y_N):$$
$$(y_1,\ldots,y_N\in Y_k)\wedge(y'\notin\{y_1,\ldots,y_N\})\wedge(\mathrm{Ver}(z,y',w')\text{=Yes})] < \mathbf{negl}(\lambda)$$

其中：概率取决于 Gen、Eval 和 $\mathcal{A}$ 的随机输出。如果对任何正整数 N，一个累加器机制都是 N 次无冲突的，那么它是无冲突的。

具有无冲突性的累加器被称为无冲突累加器，可防止攻击者通过构建虚假成员集合来伪造见证。

1.4 动态累加器

在成员认证方面的应用，要求所选择的密码累加器不仅能使验证者进行高效的身份认证，且保证安全性，当成员集合发

生变化时，累计值和各成员的见证也能以高效率更新,特别是高效成员撤销[17]；否则，每当添加或减少成员时，所有成员都需要重新计算一次当前累计值和各自的见证，累加器在成员集合动态变化的情况下，无法高效运作来应对实际应用需求。于是引入动态累加器的定义[9]，在原有四元组基础上添加增、删、更新的操作：

定义 6　动态累加器运行机制。

一个动态累加器的运行机制[10, 12]是一个包含 7 个多项式时间算法(Gen, Eval, Wit, Ver, Add, Del, Upd)的七元组,其中：

——Gen, Eval, Wit, Ver 同定义 4，属于基本架构的内容。

——Add，元素添加算法，通常是确定性算法。输入一个累加器密钥 k，一个少于 N 个元素的集合 L 的累计值 z，其中 $L \subseteq Y_k, z \in Z_k$，以及要新增的值 $y' \in Y_k$；返回集合 $L \cup \{y'\}$ 的新累计值 z'，y' 的见证 $w' \in \mathcal{W}_k$，和 Upd 算法所需的更新信息 $\mathrm{aux}_{\mathrm{Add}}$。

——Del，元素删除算法，通常是确定性算法。输入一个累加器密钥 k，一个少于 N 个元素的集合 L 的累计值 z，其中 $L \subseteq Y_k, z \in Z_k$，以及要删除的值 $y' \in L$；返回集合 $L \setminus \{y'\}$ 的新累计值 z'，和 Upd 算法所需的更新信息 $\mathrm{aux}_{\mathrm{Del}}$。

——Upd，见证更新算法，是确定性算法，用于在对 L 中元素进行添加或删除操作后，更新集合中原有的每个元素 $y \in Y_k$ 的见证 $w \in \mathcal{W}_k$。 Upd 接收 $(k, y, w, \mathrm{op}, \mathrm{aux}_{\mathrm{op}})$ 作为输入，其中 op 为 Add 或 Del；返回一个更新后的见证 w'，用于证明 y 被累计入 z'。

1.5　特征与对比

首先介绍密码累加器的几个重要特点，作为评价及选用的衡量标准。

动态性：密码累加器具备高效的元素增删和见证更新算法，详见定义 6。

强健性：密码累加器的管理员无需可信，限门信息无法被用来伪造见证。

普遍性：密码累加器不仅可提供成员见证，也可提供非成员证明。

安全假设：在安全假设声明的前提下，密码累计器的成员验证功能不受攻击者影响。

紧致性：密码累加器能将一个大集合映射到数量级较小的累计值，具体表现为累加值和见证的所需存储空间小，以及更新算法的时间复杂度低。

一些密码累加器的性能与特点如表 1 所示（其中 n 为集合中元素个数）。

表 1　各类密码累加器特点

密码累加器	动态性	强健性	普遍性	安全假设	见证/累加器体积
BeMa[8]	×	√	×	强 RSA 假设+随机谕言	O(1)
BarPif[11]	×	√	×	强 RSA 假设	O(1)
CamLys[12]	√	×	×	强 RSA 假设	O(1)
LLX[15]	√	×	√	强 RSA 假设	O(1)
AWSM[16]	√	×	×	pairings	O(1)
CHKO[14]	√	√	√	抗碰撞哈希	O(log(n))

2　基于密码累加器的认证方法

2.1　去中心化成员验证

单向累加器 h 的拟交换性保证了选定了一个初始值 $x \in X$ 后，对于集合中的值 $y_1, y_2, \ldots, y_m \in Y$，累加哈希值

$$z = h(h(h(\cdots h(h(h(x, y_1), y_2), y_3), \cdots, y_{m-2}), y_{m-1}), y_m)$$

不随计算时 y_i 的顺序而发生改变，即对上述累加值 z 和 Y 的一个任意排列 $\{y_{i_j}\}$,仍有

$$z = h(h(h(\cdots h(h(h(x, y_{i_1}), y_{i_2}), y_{i_3}), \cdots, y_{i_{m-2}}), y_{i_{m-1}}), y_{i_m})$$

此外，h 的单向性保证了已知 $x \in X$ 和 $y \in Y$，且给定 $y' \in Y$，找出一个 $x' \in X$ 使得 $h(x, y) = h(x', y')$ 的概率可忽略不计。

由此累加器可用于成员验证，同上，已知一个集合中的值 $y_1, y_2, \ldots, y_m \in Y$，则可计算所有元素的累加哈希值 z。手执 y_j 的集合成员可以计算其他所有 $y_i (i \neq j)$ 的累加值 z_j。

所以，通过出示 z_j, y_j，该成员可以向他人证明自己处于集合内，对方则可验证，若有 $z = h(z_j, y_j)$，则 $y_j \in Y$，且 z_j 称做 y_j 的见证。若非集合成员想伪造成员身份 y'，则其必须构造一个 x' 满足 $z = h(x', y')$，而这在一定假设条件下是几乎无法实现的。

单向累加器的这种使用方法并不能隐藏用户的个人信息 y_i，因为所有 y_i 都必须用来计算累加值 z，但减轻了用户负担，无需维护成员列表，从而显著节省了存储空间。并且区别于传统设立一个中心机构维护一张所有成员的列表，累加器的使用可以让用户间互相证明身份，无需某中心参与。

2.2　设计基于密码累加器的身份-公钥对认证方式

传统通信中 A 向 B 证明身份的步骤如下：

a)A 向 B 发送 (id_A, pk_A)，宣称公钥 pk_A 属于身份 id_A；

b)B 从区块链（或权威第三方）查询得到匹配的登记信息 (id_A, pk_A)，证明公钥确实属于该身份；

c)B 向 A 发送随机挑战字符串 ch，A 对包含该字符串的信进行数字签名 $\sigma = \mathrm{sig}(sk_A, \mathrm{ch})$；

d)B 用 pk_A 对数字签名 σ 进行验证，若 $\mathrm{ver}(pk_A, \sigma, \mathrm{ch}) = 1$，则证明 A 持有私钥 sk_A，即 A 拥有身份 id_A。

然而在基于区块链的去中心化身份认证体制中，对应于 B 的不同节点状态，密码累加器的引入发挥不同的作用。若 B 为完整节点，则必须查询所有本地存储的链上信息，遍历整条区块链，认证效率随着区块链体积增大不断降低；若 B 为轻节点，则没有足够的空间来存储完整区块链，无法正常进行通信认证，除非向完整节点请求查询，而这将再次面临遍历区块链效率低下的问题。

本文将步骤 a)和 b)改进为：A 向 B 发送 (id_A, pk_A, w_A)，B 进行定义 4 中的验证算法，通过运算：

$$\mathrm{Ver}(k, h(id_A, pk_A), w_A, z) = 0 \text{ or } 1$$

判断 pk_A 是否属于 id_A。此方法通常能将步骤 b)中的单步验证时间复杂度从 O(n)降至 O(log(n))或 O(1)。同时，利用密码累加器，只需存储累加值等信息，即可同样实现身份认证的功能，将本地存储的空间复杂度 O(n)降至 O(log(n))或 O(1)，使得轻节点能

正常加入认证网络参与工作。

2.3 区块链上的累加器运行方法

结合现有的基于区块链的去中心化身份认证体制以及密码累加器运行机制，本文引入累加器以提高身份认证效率和解决轻节点本地存储空间限制问题，相较于普通区块链，在区块的交易信息中添加“当前累加值”这一条目，并且每个参与节点本地维护自身见证。

本节主要描述创世区块开始累加器的创建方法，以及身份注册、废除、更新、验证过程中区块链上累加器的运行方法。

2.3.1 累加器初始创建

矿工节点创建安全参数：$k \leftarrow \mathrm{Gen}(1^{\lambda},\mathrm{N})$，并输入一张含 m 个元素的成员列表($m \ge 1$ ，至少含该矿工本身),列表元素为 (id_i, pk_i) 二元组。通过 $(z_0, \mathrm{aux}) \leftarrow \mathrm{Eval}(k, y_1, y_2, ..., y_m)$ 得到初始累加值，将其放入“创世区块”，并向全网广播。这里 $y_i = hash(id_i, pk_i)$ 。

所有接收者验证 z_0 的创建是否正确。若正确，同步该区块并向邻节点传播，并计算各自见证 $w_i \leftarrow \mathrm{Wit}(k, y_i, \mathrm{aux})$ ；否则，抛弃该区块。算法描述如算法 1 所示。

算法 1 累加器初始创建

for creator:
k=Gen(1^{λ},N)
$z_0, \mathrm{aux} = \mathrm{Eval}(k, y_1, y_2, ..., y_m)$
for i in range(len(initial_idlist)):
initial_idlist= $(id_1, id_2, ..., id_m)$
initial_pklist= $(pk_1, pk_2, ..., pk_m)$
current_transaction[i][‘identity’]= initial_idlist[i]
current_transaction[i][‘publickey’]= initial_pklist[i]
current_transaction[i][‘accu’]= z_0
genesis_block.add(current_transaction)
proof=PoW(genesis_block_header)
genesis_block.add(proof)
$w_1 = \mathrm{Wit}(k, y_1, \mathrm{aux})$ #计算自身见证,设其编号为 1
blockchain.append(genesis_block)
broadcast genesis_block
for neighbor_i in node.neighbors:
when receive genesis_block from node:
if current_block is constructed correctly:
#该区块加入本地区块链
blockchain.append(genesis_block)
#根据接收区块中新增身份更新自身见证
for (id, pk) in genesis_block:
w_i^{new} =Upd$(k, (id, pk), w_i^{old}, \mathrm{Add}, \mathrm{aux}_{\mathrm{Add}})$
else:
abort genesis_block

2.3.2 身份 id 注册

当矿工通过某节点的身份注册申请，则其将 (id, pk) 加入累加器，计算 $(z_{\mathrm{new}}, w, \mathrm{aux}_{\mathrm{Add}}) \leftarrow \mathrm{Add}(k, z_{\mathrm{old}}, (id, pk))$ ，从而得到新累计值 z 以及身份 id 的对应见证 w 。最后，矿工将信息 (id, pk, a, w) 加入新挖的区块。

所有区块接收者验证 z 计算是否正确。若是，则将自身存储区块链上的累计值更新为 z ，并通过 $w_i' \leftarrow \mathrm{Upd}(k, (id, pk), w_i, \mathrm{Add}, \mathrm{aux}_{\mathrm{Add}})$ 更新自身存储的见证；否则，抛弃该区块。算法描述如算法 2 所示

算法 2 身份注册

for miner in miners:
when receive $(id, pk, \mathrm{sig}(sk, id), \mathrm{register})$:
current_transaction=[]
if (id, pk) is well constructed:
current_transaction.add((id, pk))
#生成新累加值 z_{new} 和该注册 id 的见证 w
$(z_{\mathrm{new}}, w, \mathrm{aux}_{\mathrm{Add}}) = \mathrm{Add}(k, z_{\mathrm{old}}, (id, pk))$
current_transaction.add(z_{new})
current_block.add(current_transaction)
proof=PoW(current_block_header)
current_block.add(proof)
#更新自身见证，设其编号为 j
w_j^{new} =Upd$(k, (id, pk), w_j^{old}, \mathrm{Add}, \mathrm{aux}_{\mathrm{Add}})$
blockchain.append(current_block)
broadcast current_block
for neighbor_i in node.neighbors:
when receive current_block from node:
if current_block is constructed correctly:
blockchain.append(current_block)
w_i^{new} =Upd$(k, (id, pk), w_i^{old}, \mathrm{Add}, \mathrm{aux}_{\mathrm{Add}})$
else:
abort current_block

2.3.3 身份 id 废除

矿工通过 $\mathrm{Ver}(k, (\mathrm{id,pk}), w, z)$=1 验证该身份在累加器中。若验证正确，将该节点身份从累加器中移除，重新计算累计值 $(z_{\mathrm{new}}, \mathrm{aux}_{\mathrm{Del}}) \leftarrow \mathrm{Del}(k, z_{\mathrm{old}}, (\mathrm{id,pk}))$ ，并将 z' 加入新挖的区块；若验证失败，则中止。

所有区块接收者验证 z' 计算是否正确。若是，则将自身存储区块链上的累计值更新为 z ，并通过 $w_i'' \leftarrow \mathrm{Upd}(k, (\mathrm{id,pk}), w_i', \mathrm{Del}, \mathrm{aux}_{\mathrm{Del}})$ 更新自身存储的见证；否则，抛弃该区块。

算法 3 身份废除

for miner in miners:
when receive $(id, pk, w, \mathrm{sig}(sk, id), \mathrm{revoke})$:
current_transaction=[]
#验证通过说明该身份确实已注册
#且发布废除命令的用户拥有对应私钥
if $\mathrm{Ver}(k, (id, pk), w, z)$=1 and $\mathrm{Ver}(\mathrm{sig}(sk, id), pk, id)$=1 :
current_transaction.add($(id, pk, \mathrm{revoked})$)
#生成新累加值 z_{new} 和该注册 id 的见证 w
$z_{\mathrm{new}}, \mathrm{aux}_{\mathrm{Del}} = \mathrm{Del}(k, z_{\mathrm{old}}, (id, pk))$
current_transaction.add(z_{new})
current_block.add(current_transaction)
proof=PoW(current_block_header)
current_block.add(proof)

```
            #更新自身见证，设其编号为 j
            w_j^new =Upd(k,(id,pk),w_j^old,Del,aux_Del)
            blockchain.append(current_block)
            broadcast current_block
        else:
            abort request  (id,pk,w,sig(sk,id),revoke)
for neighbor_i in node.neighbors:
    when receive current_block from node:
        if current_block is constructed correctly:
            blockchain.append(current_block)
            w_i^new =Upd(k,(id,pk),w_i^old,Del,aux_Del)
        else:
            abort current_block
```

2.3.4 公钥更新 $pk \to pk'$

等价于分别进行身份废除和注册这两步。

用户出示见证 w，矿工根据算法 3 将 (id,pk) 废除，从累加器中移除。

用户提供新公钥 pk'，矿工根据算法 2 将 (id,pk′) 注册，加入累加器。

2.3.5 身份验证

给定身份公钥和见证 (id, pk, w)，任何用户都可通过运行验证算法 $\mathrm{Ver}(k,(\mathrm{id},\mathrm{pk}),w,z)=0$ or 1 判断该身份是否合法。

3　实验

3.1　区块构建

本文采用文献[8,9]中的强 RSA 安全假设的密码累加器，基于区块链生成方法修改区块构建代码，部分代码如下，所得区块结构如图 1 所示。

```
def new_block(self, proof, pre_hash):
  block = {
    'index': len(self.chain) + 1,
    'timestamp': time.time(),
    'transactions': self.current_transactions,
    'proof': proof,
    'pre_hash': pre_hash or self.hash(self.chain[-1]),
  }
  #update accumulated value and witness
  if len(self.chain) > 0:
    idpkbind = self.current_transactions[0]['identity'] + \
      self.current_transactions[0]['publickey']
    idpkhash = self.hash(idpkbind)
    idpkconvert = int(idpkhash, 16)
    for x in range(len(self.chain)):
      self.witness[x] = pow(self.witness[x], idpkconvert, \ self.n)
    self.witness.append(self.accu)
    self.accu = pow(self.accu, idpkconvert, self.n)
  # Reset the current list of transactions
  self.current_transactions = []
  self.chain.append(block)
  return block
```

块高度：390610
头哈希：00000000002c8…ae5
父哈希：00000000003f2…f1d
Merkle根：c8572f19112…456d
时间戳：2018-3-1 10:35:12
难度：93448670796.32380676
Nonce：1779633802
Identity：mercer4524qs
Public key：Xup1R0c…JFJ83
Accu：cbf53bef6ea6d4…728f
…………

块高度：390609
头哈希：00000000003f2…f1d
父哈希：00000000005e1…e25
Merkle根：c59e2d8242…ef1c
时间戳：2018-3-1 10:25:07
难度：93448670796.32380676
Nonce：4005489007
Identity：8sJk3jcLs1df3
Public key：sraz362…JV34A3
Accu：b528c06158b6df…7878
…………

块高度：390608
头哈希：00000000005e1…e25
父哈希：000000000079f…e4d
Merkle根：2e11abce579…e12a
时间戳：2018-3-1 10:21:36
难度：93448670796.32380676
Nonce：2181060612
Identity：Lilygancelor
Public key：fVF9U63s…76tl
Accu：931143eecaf47f…e461
…………

图 1　区块结构

3.2　结果与分析

本文以三台配置 Windows7 64 位系统，Intel® Xeon(R) CPU E7520 @1.86 GHz，内存 DDR3 16 GB 的 PC 作为实验基础。

实验先建立了 5 000~30 000 条，步长 5 000 的六组身份—公钥对数据集，其中身份由 6~20 位的数字和大小写字母随机组成；公钥由 15 位数字和大小写字母随机组成；强 RSA 密码累加器门限 N 为十进制 150 位的正整数，是两个通过 Miller-Rabin 素性检测算法随机生成的强素数的乘积；种子 x 随机取一个不大于 N 的正整数。

接下来在六组数据集上，实验遍历区块链查询的传统认证方法和引入密码累加器的认证方法的耗时对比，如图 2 所示。

由图 2 可以看出，传统方法中遍历区块链的时间与区块链长度成线性关系，而利用强 RSA 安全假设的累加器进行验证与区块链长度无关，耗时仅为累加器单步验证运算的时间，在某常数的小范围内波动，此为不同运算时刻机器状况差异所致。

然而在实际情况下，密码累加器的更新和维护需额外增加计算量。本文考虑两种方法：

a)挖矿节点负责维护额外维护所有成员见证。

b)完整节点只需维护图 1 中的区块结构，挖矿节点无需计算除自身外其他节点见证，各节点自行依照区块内容计算更新自身见证。并且非随时在线的用户，在离线时记录当前区块高度 i，当下次登录时向其他完整节点请求同步序号 i 至最新的区块，并更新自身见证。

上述两种情况与传统无累加器区块链的生成速度如图 3 所示。

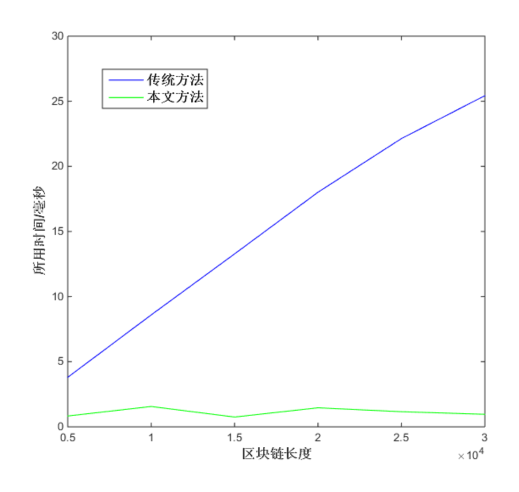


图 2　单步验证时间对比

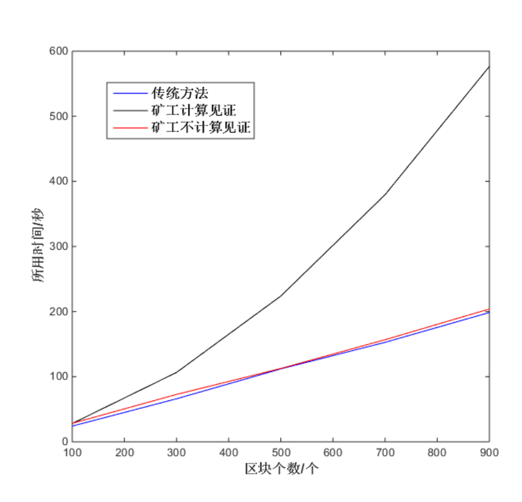


图 3　不同方法下构建区块链所需时间对比

图 3 说明了引入累加器后，若需矿工维护所有节点见证，则耗时以非线性增长，设当前区块链长度为 n，则矿工须比不维护其他节点见证时多做 $1+2+\cdots+(n-1)=\frac{n(n-1)}{2}$ 次见证更新运算。而无需计算其它节点见证的情况下，各节点每生成一个区块，比传统方法多做一步累加器求值运算，耗时曲线与传统方法接近重合，说明各节点维护自身见证，不会影响完整节点的计算量。最后，在实际情况中，采用至少 1 024 bit 的模数保证强 RSA 假设下累加器的安全性，需要实验证明模数规模较大的情况下仍能保证系统效率。

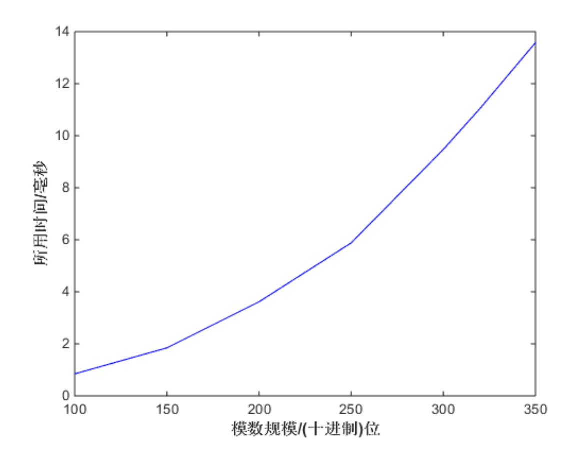


图 4　不同规模模数下的单步认证时间

图 4 说明了随着模数变大，用户的单步认证时间成非线性增长，但当模超过 1 024 bit（约十进制 308 位）时，仍为毫秒级，对于普通用户处于完全可接受的范围。

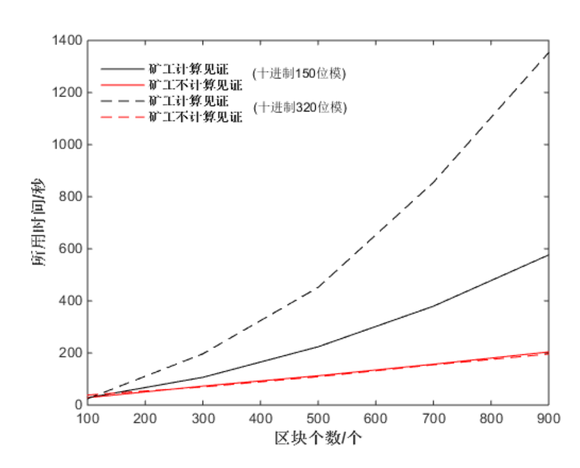


图 5　不同模数下两种方法的区块链构建时间

将 N 改为符合前文条件的十进制 320 位正整数，再次进行区块链构建实验，并将结果与之前对比。如图 5 所示，采用方法 1 时，随着模数变大，区块链构建时间明显增大；而采用方法 2 时，矿工每构建一个区块，只需做更新累加值的运算，因模数变大而增加的计算量只会增加毫秒级的时间耗费，区块链构建时间与之前基本相同。

由以上结果可见，本文引入密码累加器，并采用节点更新维护自身见证的认证方法中，用户的认证效率相较传统方法得到提升，并随区块链长度增加而不断提高；矿工在构建区块时的计算量相较传统方法增加不大，与 PoW 计算量相比可忽略不计；并且在为保证安全性增大模数时，系统仍具有很好的可用性。

4　结束语

近几年来比特币和各种竞争币受到了越来越多关注，而大多停留在数字货币的层面，作为底层技术，区块链的落地应用并不多。本文紧跟技术前沿，研究区块链在去中心化 PKI 上的应用并加以改进。首先深入分析研究了密码累加器的定义和作用原理，分析比较了主流累加器在不同评判标准下的特点。进

一步引入密码累加器，尝试改进身份—公钥认证环节的效率，从而降低整个认证过程的时间和空间复杂度，根据不同累加器，可将传统身份—公钥查询和验证过程的时间复杂度从 O(n)降为 O(log(n))甚至 O(1)。最后通过实验，验证了引入强 RSA 假设的累加器后，身份—公钥对验证环节耗时为常数，密码累加器的运行算法具有正确性和有效性，大大提升了去中心化 PKI 在链上数据庞大时的验证效率。

由于现有的强 RSA 假设累加器不具备强健性，在不存在可信管理者时有一定局限性，无法保证限门信息不被用于进行强制删除用户等不合法操作。在后续研究中可加以改进，采用基于抗碰撞哈希函数的类 Merkle Tree 型等累加器，使系统具有更好的强健性。

参考文献：

[1] Garman C, Green M, Miers I. Decentralized anonymous credentials [C]// Proc of Network and Distributed System Security Symposium. 2014.

[2] Kulynych B, Isaakidis M, Troncoso C, *et al.* ClaimChain: decentralized public key infrastructure [J]. arXiv preprint arXiv: 1707. 06279, 2017.

[3] Anada H, Kawamoto J, Weng Jian, *et al.* Identity-embedding method for decentralized public-key infrastructure [C]// Proc of International Conference on Trusted Systems. Cham: Springer, 2014: 1-14.

[4] Morselli R, Bhattacharjee B, Katz J, *et al.* Keychains: a decentralized public-key infrastructure [R]. [S. l.] : University of Maryland, College Park College Park United States, 2006.

[5] Nakamoto S. Bitcoin: a peer-to-peer electronic cash system [J]. Consulted, 2008.

[6] 袁勇，王飞跃. 区块链技术发展现状与展望 [J]. 自动化学报，2016, 42 (4): 481-494. (Yuan Yong, Wang Feiyue. Blockchain: the state of the art and future trends [J]. ACTA automatica Sinica, 2016, 42 (4): 481-494.)

[7] Miers I, Garman C, Green M, *et al.* Zerocoin: anonymous distributed e-cash from bitcoin [C]// Proc of IEEE Symposium on Security and Privacy. 2013: 397-411.

[8] Benaloh J, De Mare M. One-way accumulators: a decentralized alternative to digital signatures [C]// Proc of Workshop on the Theory and Application of of Cryptographic Techniques. Berlin: Springer, 1993: 274-285.

[9] Nyberg K. Commutativity in cryptography [C]// Proc of the 1st International Workshop on Functional Analysis at Trier University. [S. l.] : Walter de Gruyter & Co, 1996: 331-342.

[10] Fazio N, Nicolosi A. Cryptographic accumulators: definitions, constructions and applications [J]. Paper written for course at New York University: www. cs. nyu. edu//nicolosi//papers//accumulators. pdf, 2002.

[11] Barić N, Pfitzmann B. Collision-free accumulators and fail-stop signature schemes without trees [C]// Proc of International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 1997: 480-494.

[12] Camenisch J, Lysyanskaya A. Dynamic accumulators and application to efficient revocation of anonymous credentials [C]// Proc of Annual International Cryptology Conference. Berlin: Springer, 2002: 61-76.

[13] 万国根，周世杰，秦志光. 单向累计函数技术分析 [J]. 计算机科学，2005, 32 (8): 57-59. (Wan Guogen, Zhou Shijie, Qin Zhiguang. An overview of the one-way accumulator technology [J]. Computer Science, 2005, 32 (8): 57-59.)

[14] Camacho P, Hevia A, Kiwi M, *et al.* Strong accumulators from collision-resistant hashing [C]// Proc of International Conference on Information Security. Berlin: Springer, 2008: 471-486.

[15] Li Jiangtao, Li Ninghui, Xue Rui. Universal accumulators with efficient nonmembership proofs [C]// Proc of Applied Cryptography and Network Security. Berlin: Springer, 2007: 253-269.

[16] Au M H, Wu Qianhong, Susilo W, *et al.* Compact e-cash from bounded accumulator [C]// Proc of Cryptographers'Track at the RSA Conference. Berlin: Springer, 2007: 178-195.

[17] 陈泽文，王继林，黄继武，等. ACJT 群签名方案中成员撤消的高效实现 [J]. 软件学报，2005, 16 (1): 151-157. (Chen Zewen, Wang Jilin, Huang Jiwu, *et al.* An efficient revocation algorithm in ACJT group signature [J]. Journal of Software, 2005, 17 (1): 33-50.)

[18] 邵奇峰，金澈清，张召，等. 区块链技术：架构及进展 [J]. 计算机学报，2017: 1-20. (Shao Qifeng, Jin Cheqing, Zhang Zhao, *et al.* Blockchain technology: architecture and progress [J]. Chinese Journal of Computers, 2017: 1-20.)

[19] Sasson E B, Chiesa A, Garman C, *et al.* Zerocash: decentralized anonymous payments from bitcoin [C]// Proc of IEEE Symposium on Security and Privacy. 2014: 459-474.